



Merlin ToolChain

Enabling transparency beyond partner borders

Rob Kommeren - Philips

2007-09-25

Content

1. Background
2. Merlin Toolchain
 - Goals
 - Architecture
 - Implementation
3. Implementation – The Demonstrator
4. Implementation – Philips OSIB project
5. Conclusions, current and future work

Background

- With small scale product development of the past, only few tools were used.
- Nowadays software development is becoming increasingly complex and productivity requirements are so high that development tools are indispensable.
- No tool or development entity is an island. Instead, they are connected and these connections need to be managed efficiently.
- Interoperability is not readily available in development tools. Data transfer between tools used in different SW development phases is considered difficult.
- Tools from different vendors do not share common data formats. As a result manual data transferring and handling is needed
 - Manual data transfer is laborious and increases errors.
- The impact of the issues stated above is even higher in view of the trend to collaborative software development...

Collaborative Perspective

- Companies rarely have the same sets of software tools. Typical consequent problems are:
 - Poor coordination and synchronisation of work flow
 - Poorly alignment of work products and their statuses
 - Lack of transparency of development for parties involved

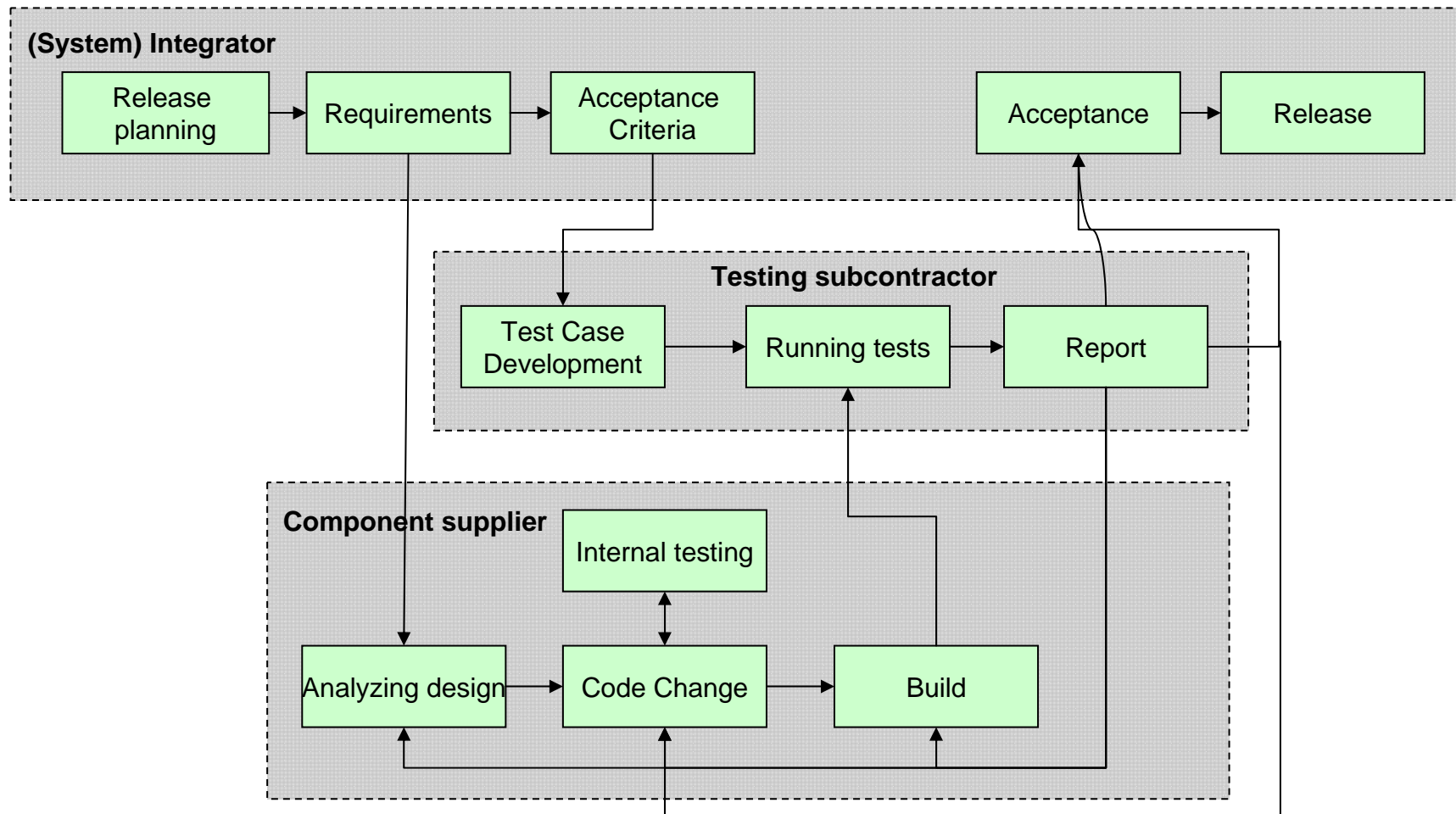
... Resulting in an inefficient and ineffective development process
- Constraint:
Industrial experience shows the need for integrable rather than integrated tool solutions
 - Proven solutions for parts of the problems should be kept!

The solution: **Tool integration optimised for personal needs!**

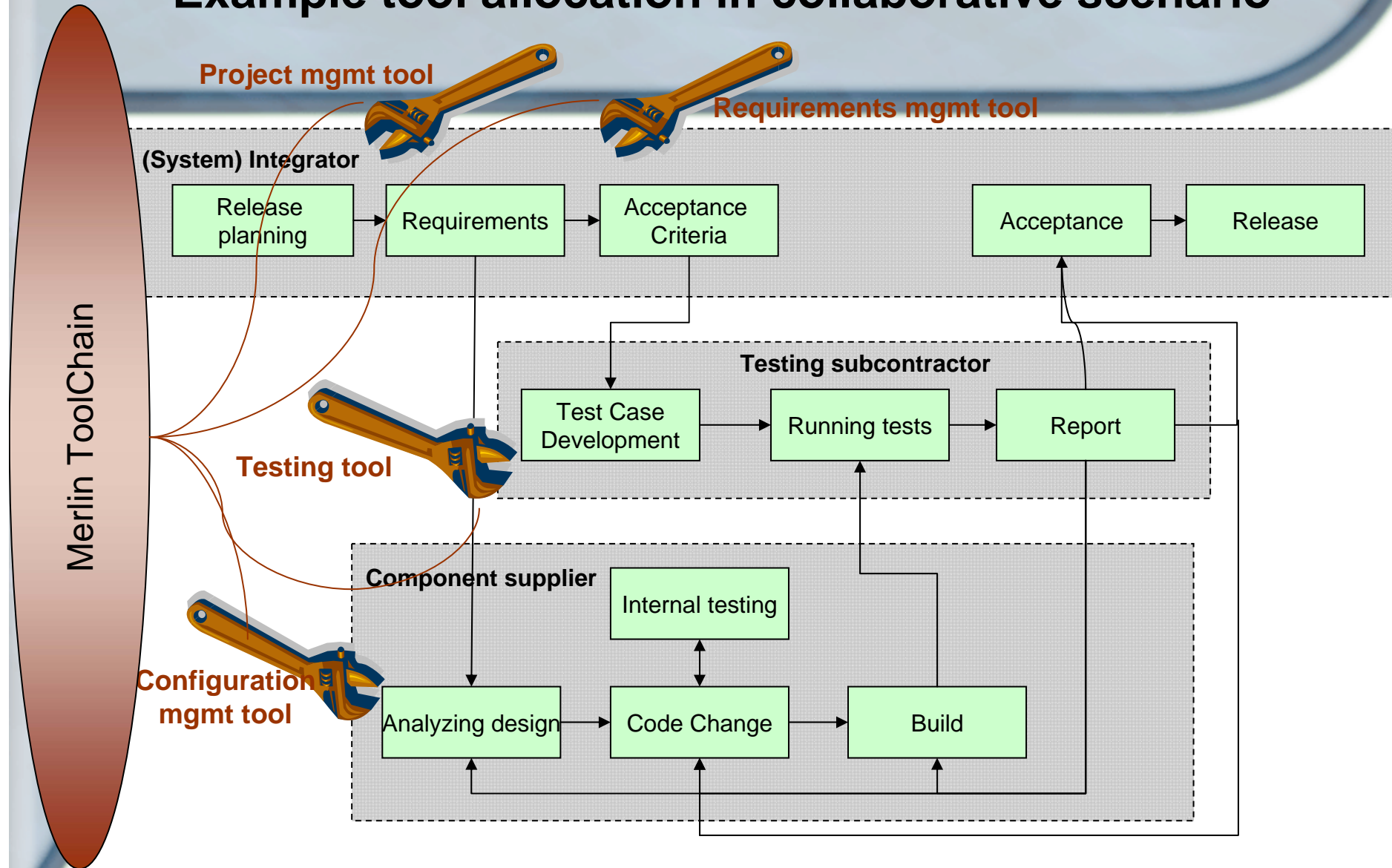
Merlin ToolChain Goals

- To support collaborative distributed SW projects by developing a concept for integrating existing tools in a meaningful way
- To build an example implementation of the Merlin Toolchain by using selected sets of tools
- To pilot the implementation both in a demonstrator and in a real life industrial project
- To gather experiences for further development

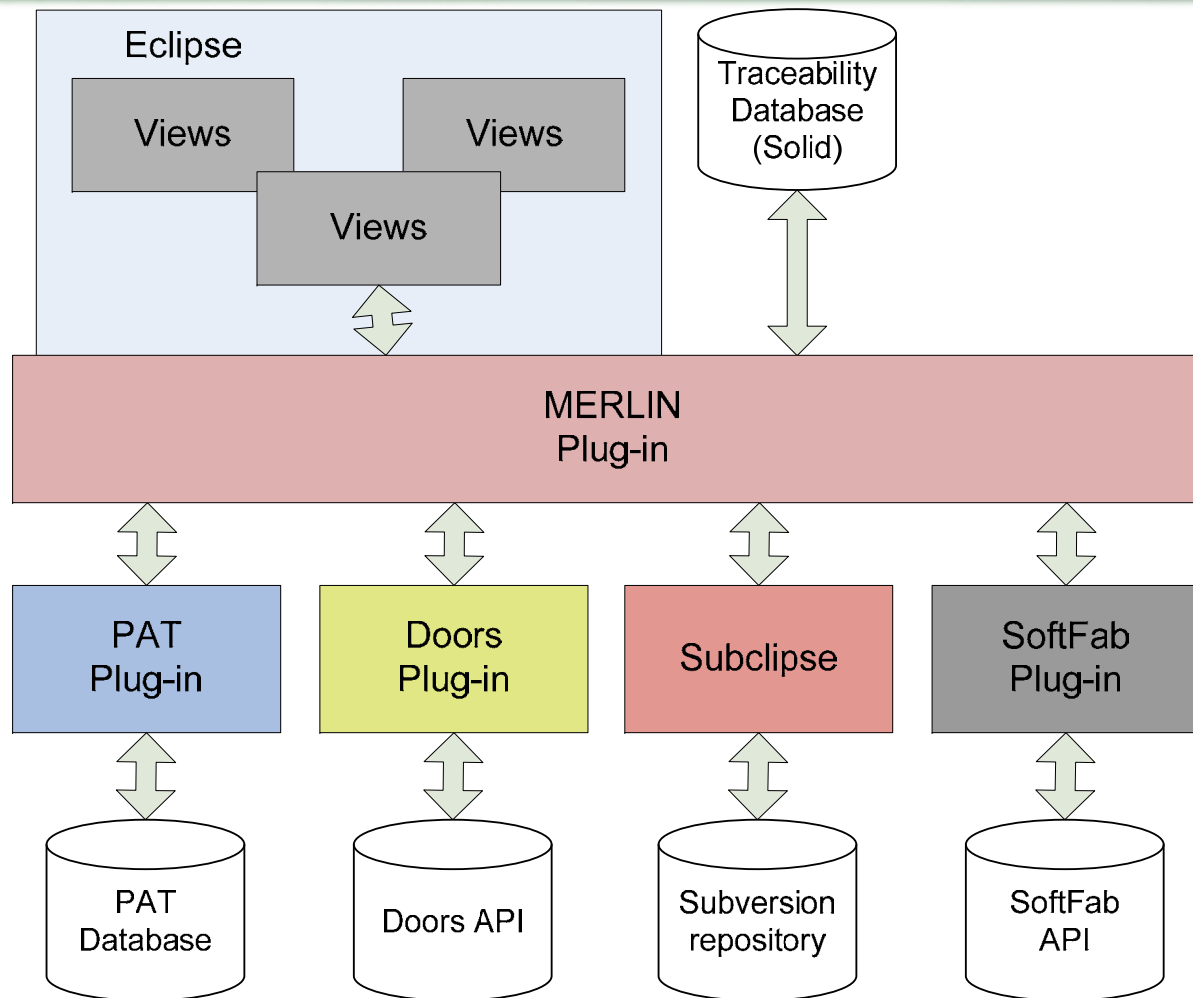
Reference: Typical collaborative scenario



Example tool allocation in collaborative scenario



Merlin ToolChain Overall Architecture

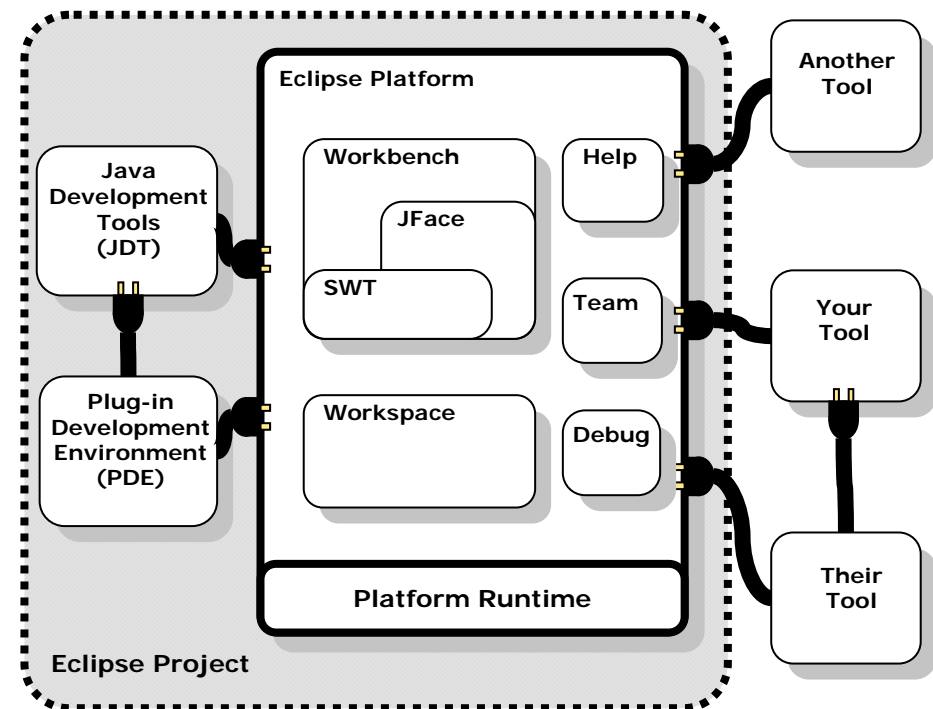


Tool Chain Implementation

- Eclipse is used as an integration framework
- The ToolChain is implemented by developing Eclipse plug-ins:
 - The main plug-in called Merlin Plug-in controls these separate plug-ins and formulates the completeness of the tool chain under the Eclipse IDE
 - Tools are accessed with own separate plug-ins. E.g.,:
 - Doors (made for Telelogic Doors tool)
 - PATclipse (made for Philips PAT tool)
 - Subclipse (Existing plug-in, integrated with ToolChain)
 - SoftFab (made for Philips SoftFab tool)

Eclipse Introduction

- The Eclipse Platform is an open source tool designed for building integrated development environments (IDEs)
- Can be used to develop e.g. Java and C++ programs, supporting also modelling, testing and embedded development
- Written in the Java language and comes with plug-in based framework that makes it easier to create, integrate and utilize software tools
- Eclipse Platform has a small kernel called as the Platform Runtime, but almost all functionality is located in plug-ins
- A plug-in is the smallest unit of Eclipse Platform function that can be developed and delivered separately



Selection criteria for tools to be incorporated in the ToolChain

- To start, for each of the following development areas one tool was selected
 - Project Management
 - Requirement Management
 - Configuration Management
 - Testing
- Selection of the was based on investigation at Merlin industrial partners
 - Tools that some industrial partners are using
 - Tools that were wanted for detailed analysis
- Tools were also selected from different vendors on purpose, to emulate the real world situation as well as possible

Selected Tools

- Initially selected tools were
 - **Philips Project Assist Tool** (Project Management)
 - **IBM Rational RequisitePro** (Requirements Management)
 - **Telelogic Synergy/CM** (Configuration Management)
 - **Philips SoftFab** (Testing)
- To prove that tools could be exchanged in the ToolChain, new tools were introduced in a later stage:
 - **RequisitePro → Telelogic Doors™**
 - **Telelogic Synergy/CM → Subversion**



Merlin ToolChain is customizable!

Use any combination or subset of the tools already plugged in

Or, plug-in your own tool according to instructions!



PHILIPS

Project Assist Tool

Project mgmt tool



SynergyCM



Subversion

MERLIN

Login Settings **Tools setup**

PM Tool	RM Tool
<input checked="" type="radio"/> Project Assist Tool	<input type="radio"/> Requisite Pro
<input type="radio"/> Open Workbench	<input checked="" type="radio"/> Doors
<input type="radio"/> Not in use	<input type="radio"/> Not in use

CM Tool	TM Tool
<input type="radio"/> Synergy/CM	<input checked="" type="radio"/> Softfab
<input checked="" type="radio"/> Subversion	<input type="radio"/> TestLink
<input type="radio"/> Not in use	<input type="radio"/> Not in use

Save settings



Doors



OSRMT



Requirements mgmt tool



SoftFab

Demonstrator case



Three partners are working in collaboration

- Integrator (Finland)
 - Requirements Management and Project management
- Supplier (USA)
 - Software implementation
- Subcontractor (China)
 - Software testing

Demonstrator Goals

- Uniform project view for all partners
- Up-to-date information available to all the partners

Demonstrator script

- Integrator fills in tasks to project mgmt tool and assigns them to subcontractor and supplier
 - Tasks can be monitored in Merlin ToolChain by all parties
- Integrator writes requirement description in RM tool and links it to tasks in the Merlin ToolChain
- Integrator fills effort estimation for the task
- Component supplier sees new task assignment and develops new code accordingly
- Component supplier records effort spent in project mgmt tool
- Testing subcontractor sees component supplier is ready with code development
- Testing subcontractor runs tests for the component
- Testing subcontractor records effort spent
- Integrator can see the test results and accepts the component accordingly

Actual implementation

Requirements mgmt

DOORS

Configuration mgmt

Subversion

Testing

SoftFab

Project mgmt

Task status

PAT

Requirements list

Source code list related to task

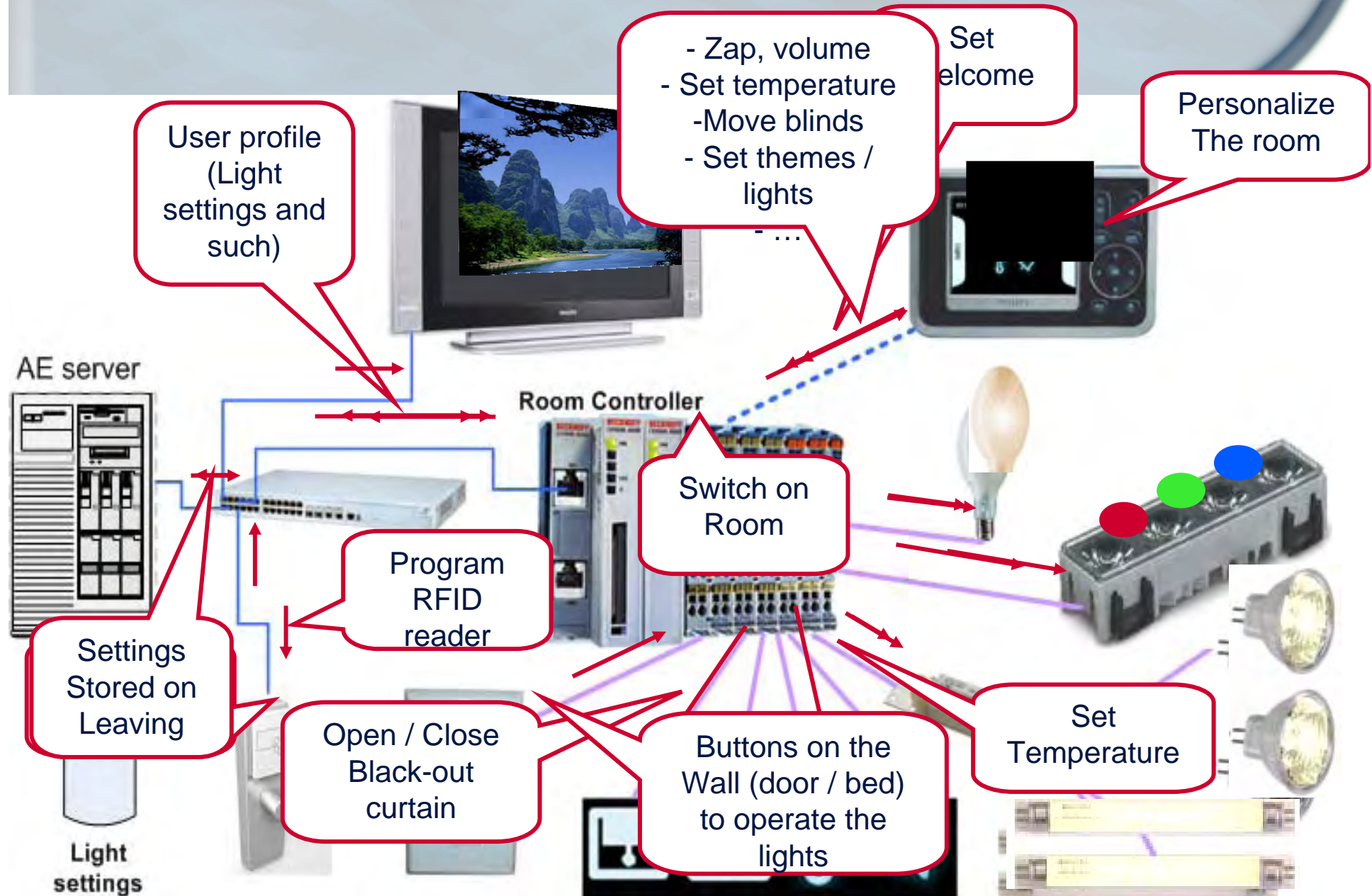
Test cases related to requirement

Industrial case: Philips OSIB project

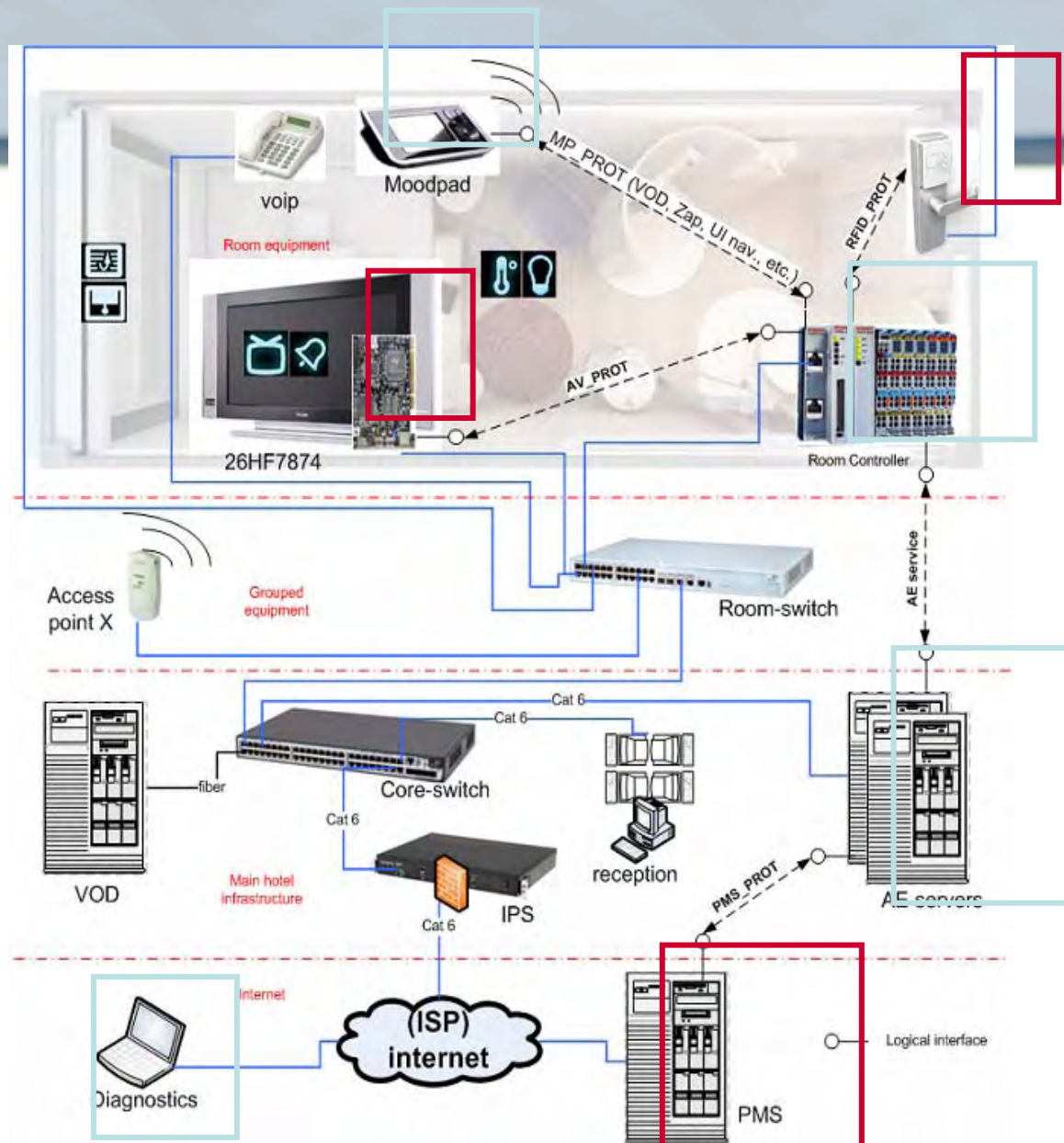
Purpose of the industrial case

- To prove that Merlin solutions work in an integrated fashion in real-life industrial projects (referring ITEA reviewers' comments)
- To implement the Merlin tool chain in an industrial project
 - Enabling smooth continuation of the day-to-day project execution
 - Targeted added value for the project:
Full transparency and traceability of
 - Project tasks
 - Requirements
 - Related source code
 - Build and test status

Merlin The OSIB project: Today's Experience



The OSIB project: Hospitality implementation



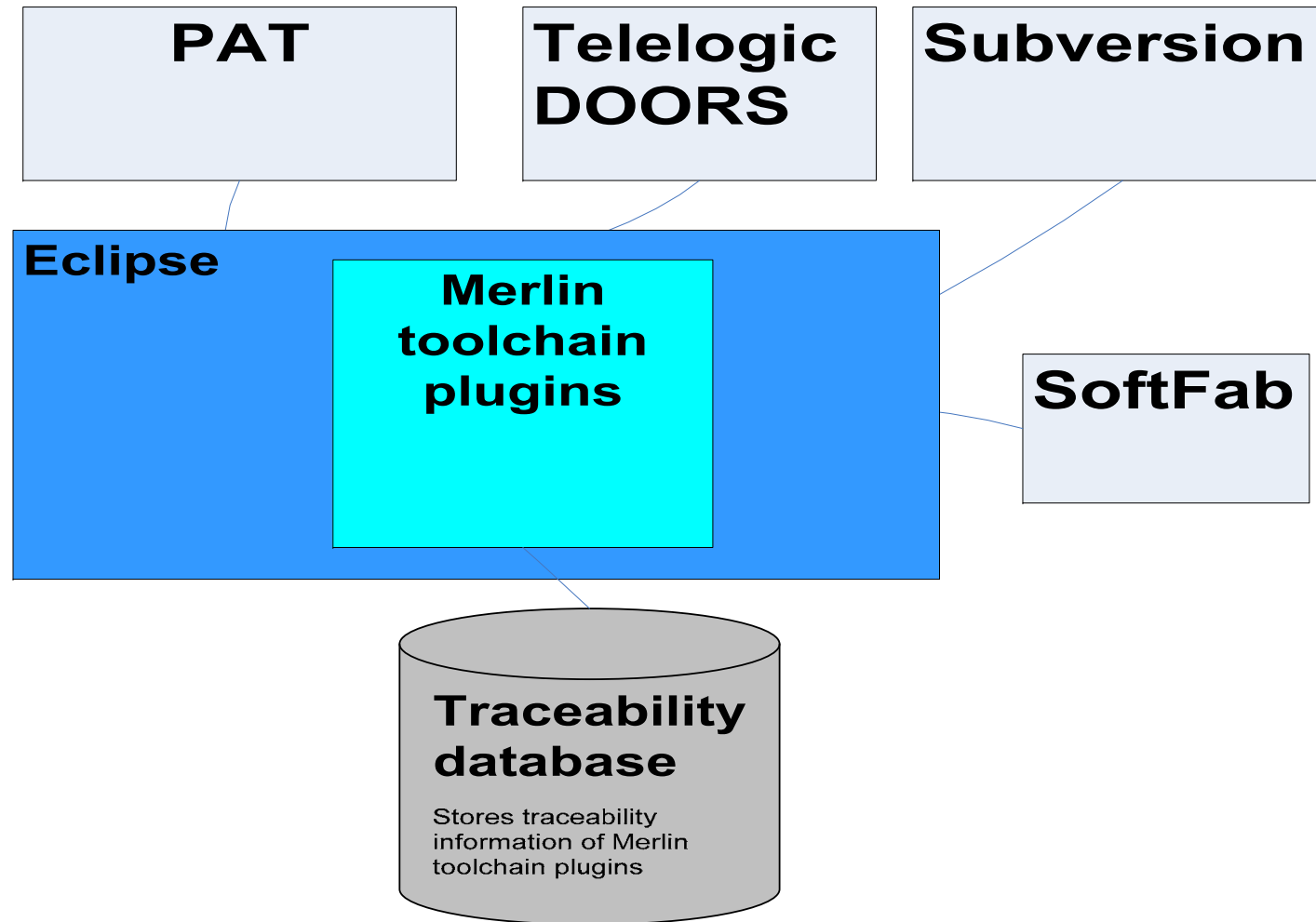
3rd party
integration
points

Software
made by
Philips

Project characteristics

- Implementation agreed with customer in phases of 3-6 months
 - Project is in 3rd implementation phase
- Team size: 6 engineers (during implementation)
- Agile way of working
 - Development in increments of two weeks
 - Requirements are selected for each increment, together with customer
- OSIB's native tool-set:
PAT, DOORS, Subversion, SoftFab

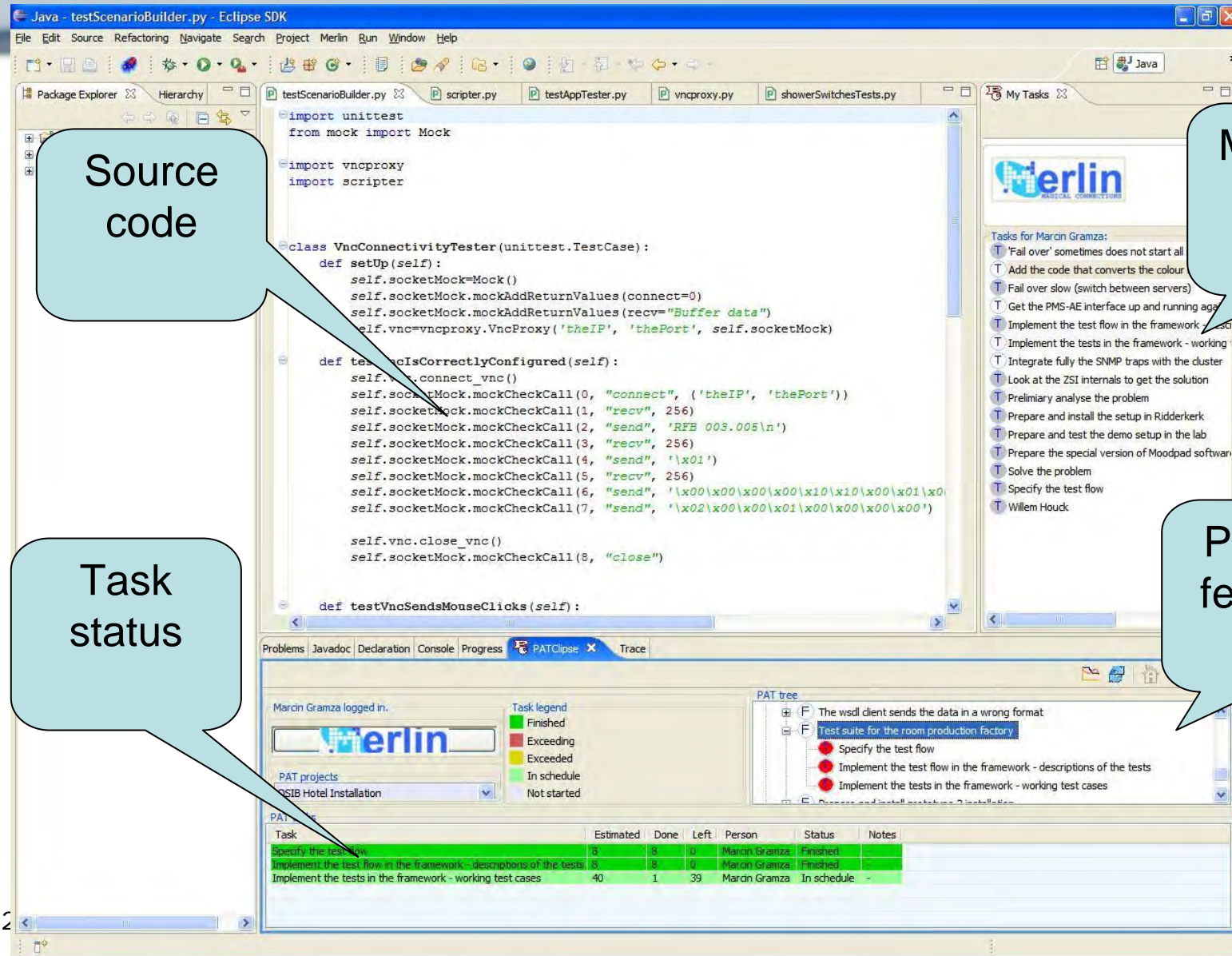
Merlin ToolChain: Instantiation for OSIB



Implementation trajectory

- | | |
|------------|---|
| Wk 710-712 | Preparation of (updates of) plug-ins by implementation team (Philips (1), VTT (2)) <ul style="list-style-type: none">- Subversion (new plug-in)- Doors (new plug-in)- SoftFab (plug-in update)- PAT (plug-in update) |
| Wk 713 | Implementation of tool chain in OSIB project, including real-life project data |
| Wk 714 | Demo to Merlin representatives + Philips project leaders |

Tool chain screen dumb: Feature-Task Overview



Source code

```

import unittest
from mock import Mock

import vncproxy
import scripiter

class VncConnectivityTester(unittest.TestCase):
    def setUp(self):
        self.socketMock=Mock()
        self.socketMock.mockAddReturnValues(connect=0)
        self.socketMock.mockAddReturnValues(recv="Buffer data")
        self.vnc=vncproxy.VncProxy('theIP', 'thePort', self.socketMock)

    def testVncIsCorrectlyConfigured(self):
        self.vnc.connect_vnc()
        self.socketMock.mockCheckCall(0, "connect", ('theIP', 'thePort'))
        self.socketMock.mockCheckCall(1, "recv", 256)
        self.socketMock.mockCheckCall(2, "send", 'RFB 003.005\n')
        self.socketMock.mockCheckCall(3, "recv", 256)
        self.socketMock.mockCheckCall(4, "send", '\x01')
        self.socketMock.mockCheckCall(5, "recv", 256)
        self.socketMock.mockCheckCall(6, "send", '\x00\x00\x00\x00\x10\x10\x00\x01\x00')
        self.socketMock.mockCheckCall(7, "send", '\x02\x00\x00\x01\x00\x00\x00\x00')

        self.vnc.close_vnc()
        self.socketMock.mockCheckCall(8, "close")

    def testVncSendsMouseClicks(self):

```

My tasks

Tasks for Marcin Gramza:

- Fail over' sometimes does not start all
- Add the code that converts the colour
- Fail over slow (switch between servers)
- Get the PMS-AE interface up and running again
- Implement the test flow in the framework - working
- Implement the tests in the framework - working
- Integrate fully the SNMP traps with the cluster
- Look at the ZSI internals to get the solution
- Preliminary analyse the problem
- Prepare and install the setup in Ridderkerk
- Prepare and test the demo setup in the lab
- Prepare the special version of Moodpad software
- Solve the problem
- Specify the test flow
- Willem Houck

Task status

Marcin Gramza logged in.

PAT projects

OSIB Hotel Installation

Task legend

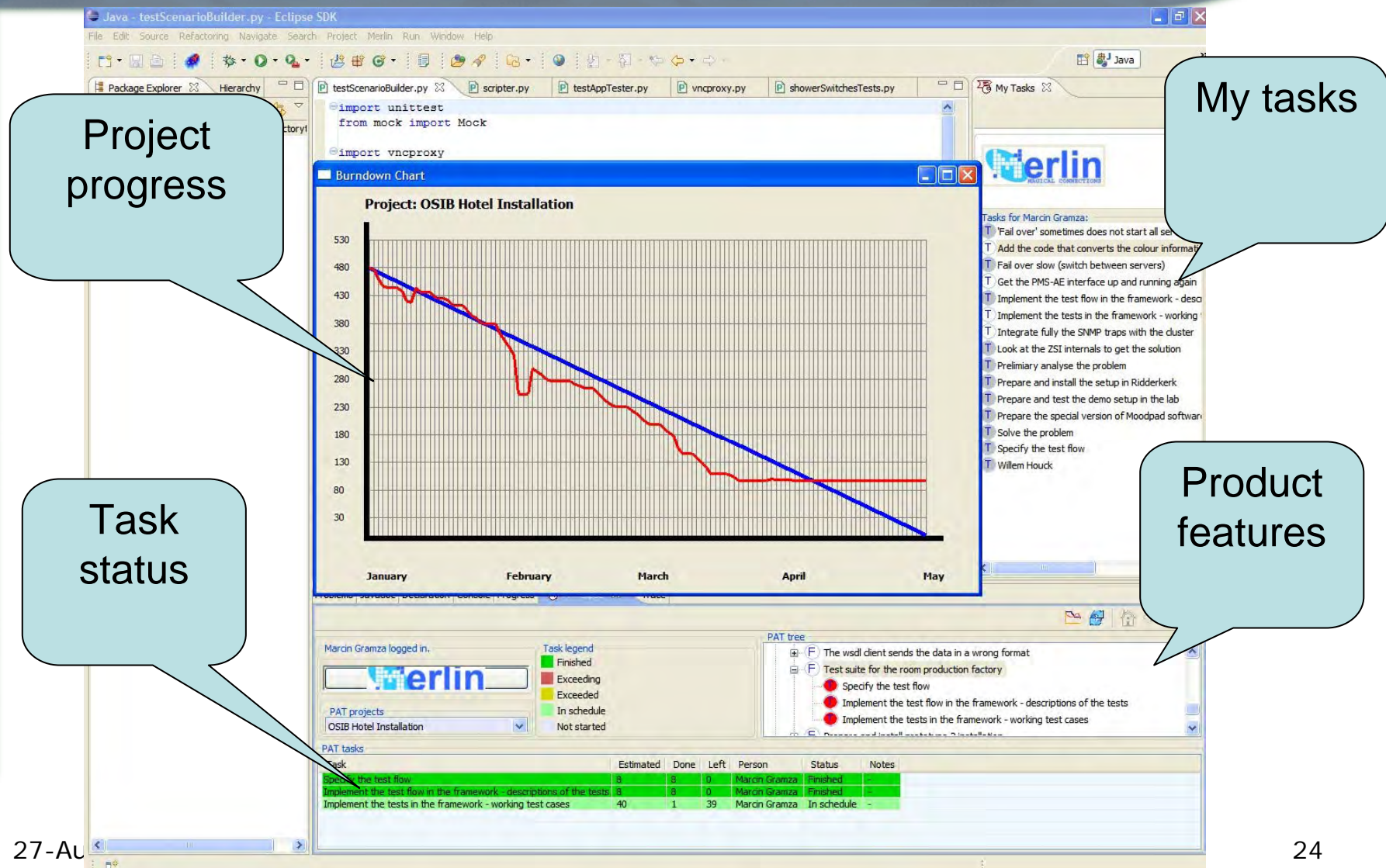
- Finished
- Exceeding
- Exceeded
- In schedule
- Not started

PAT tree

- The wsdl client sends the data in a wrong format
- Test suite for the room production factory
 - Specify the test flow
 - Implement the test flow in the framework - descriptions of the tests
 - Implement the tests in the framework - working test cases
- Source and install installation

Task	Estimated	Done	Left	Person	Status	Notes
Specify the test flow	8	8	0	Marcin Gramza	Finished	
Implement the test flow in the framework - descriptions of the tests	8	8	0	Marcin Gramza	Finished	
Implement the tests in the framework - working test cases	40	1	39	Marcin Gramza	In schedule	

Tool chain screen dumb: Task Burn-down



Traceability Overview

The screenshot shows the Eclipse IDE with the Merlin project open. The main editor displays the source code of `blindsTests.py`. The left sidebar shows the project hierarchy. The bottom of the IDE is divided into several views: **Problems**, **Trace**, **Requirements tree**, **Code files**, and **Test cases**.

Task status (Annotation pointing to the Merlin logo and task list):

The Merlin logo is visible in the **Trace** view. Below it, a list of tasks is shown with their status:

Task	Estimated	Done	Left	Person
Music genre: Implement changes in Room Controller	8	4	4	Jerzy H.
RFID log: Write description document	4	9	0	Jerzy H.
RFID log: Implement changes in Diagnostics	8	4	0	Jerzy H.

Requirements (Annotation pointing to the Requirements tree):

The **Requirements tree** view shows a hierarchy of requirements:

- Requirements
 - 1. Tablet PC
 - 1.1 The user interface
 - 1.1.1
 - 1.2 Durability
 - 1.2.1
 - 1.2.2

Source code related to task (Annotation pointing to the Code files view):

The **Code files** view shows a table of files:

File name	Version	Author	Size	Repository path
blindsTests.py	55	jpjukka	1	svn://192.168.1.2/OSIB2

Test cases related to requirements (Annotation pointing to the Test cases view):

The **Test cases** view shows a table of test cases:

Test case	Status
Durability_Test_2	ok
Durability_test1	error

OSI B Conclusions

- Easy preparation of plug-ins
- Fast and successful implementation trajectory
- Full transparency demonstrated on real-life project data, according to expectations
 - Total overview on one screen!
 - Unified user-interface for various views (tasks, requirements, code, build & test)
 - No need to get used to individual tool user-interfaces
- Project leaders enthusiastic:
"When can I have it!"

Overall Conclusions

- Merlin Tool demonstrates, that integration of tools from different vendors is possible, and it answers directly to the identified challenges
 - Efficient use of resources
 - Transparency through partner borders
 - Traceability between the tested product and the requirements
 - Common understanding of the requirements
 - Consistency in engineering tasks

Current and Future Work

- Improving and further developing the Merlin Tool
 - Adding possibility to exchange tools in the area of:
 - Requirements Management (DOORS → OSRMT)
 - Testing (SoftFab → TestLink)
 - Project management (PAT → Open Workbench)
 - Focus on incorporation of open source tools
- Status of the toolchain (e.g. PR/CR database) maintained by VTT
- Iterative evaluation
 - Each 2 weeks feedback from Philips- AppTech, by gathering opinions from users
 - Evaluation results sent to VTT
 - Minor problems / change requests have been issued related to
 - User Interface
 - Start-up sequence of tool chain
 - Unexpected behavior (e.g., missing information in some cases)
- Create a final report about the tool chain development